Journal of Computer and System Sciences 77 (2011) 1039–1053

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Almost tight upper bound for finding Fourier coefficients of bounded pseudo-Boolean functions

Sung-Soon Choi [a,1], Kyomin Jung [b,*,2], Jeong Han Kim [a,c,3]

[a] Department of Mathematics, Yonsei University, Seoul, 120-749, Republic of Korea
[b] Department of Computer Science, KAIST, Daejeon, 305-701, Republic of Korea
[c] National Institute for Mathematical Sciences, Daejeon, 305-340, Republic of Korea

ARTICLE INFO

Article history:
Received 29 December 2008
Received in revised form 9 August 2010
Accepted 19 August 2010
Available online 31 August 2010

Keywords:
Pseudo-Boolean function
Fourier coefficients
Graph finding
Learning polynomials
Linkage discovery
Walsh analysis

ABSTRACT

A k-bounded pseudo-Boolean function is a real-valued function on $\{0,1\}^n$ that can be expressed as a sum of functions depending on at most k input bits. The k-bounded functions play an important role in a number of areas including molecular biology, biophysics, and evolutionary computation. We consider the problem of finding the Fourier coefficients of k-bounded functions, or equivalently, finding the coefficients of multilinear polynomials on $\{-1,1\}^n$ of degree k or less. Given a k-bounded function f with m non-zero Fourier coefficients for constant k, we present a randomized algorithm to find the Fourier coefficients of f with high probability in $\mathcal{O}(m \log n)$ function evaluations. The best known upper bound was $\mathcal{O}(\lambda(n,m)m\log n)$, where $\lambda(n,m)$ is between $n^{\frac{1}{2}}$ and n depending on m. Our bound improves the previous bound by a factor of $\Omega(n^{\frac{1}{2}})$. It is almost tight with respect to the lower bound $\Omega(\frac{m\log n}{\log m})$. In the process, we also consider the problem of finding k-bounded hypergraphs with a certain type of queries under an oracle with one-sided error. The problem is of self interest and we give an optimal algorithm for the problem.

1. Introduction

A *pseudo-Boolean* function is a real-valued function on $\{0.1\}^n$. If a pseudo-Boolean function can be expressed as a sum of functions depending on at most k input bits, it is called *k-bounded*. Unless otherwise specified, a k-bounded function means a k-bounded pseudo-Boolean function in this paper. We consider the problem of finding the Fourier coefficients of k-bounded functions. In the problem, we assume the oracle for membership queries, i.e., the oracle that, given any binary string, returns the function value at the string. Our main concern is the query complexity to solve the problem.

By definition, degree-k polynomials on $\{0,1\}^n$ are k-bounded functions, and each k-bounded function can be represented as a polynomial of degree k or less. We will use the term 'k-bounded functions' rather than 'degree-k polynomials' as the definition of the former captures the critical property of functions for our results nearly regardless of representation.

* Corresponding author.
E-mail addresses: ss.choi@yonsei.ac.kr (S.-S. Choi), kyomin@kaist.edu (K. Jung), jehkim@nims.re.kr (J.H. Kim).

[1] This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R16-2007-075-01000-0).
[2] This work was supported by NAP of Korea Research Council of Fundamental Science and Technology.
[3] This work was partially supported by Yonsei University Research Funds 2006-1-0078 and 2007-1-0025, and by the second stage of the Brain Korea 21 Project in 2007, and by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2006-312-C00455).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.08.011

Also, notice that, by a simple change of notation, the considered problem is equivalent to that of finding the coefficients of polynomials on $\{-1,1\}^n$ of degree k or less.

The k-bounded functions have played an important role in a number of research areas. In molecular biology and biophysics, they have been used for *fitness functions* that model the environments in which biological objects are evolved [26]. The k-bounded functions with small k have been paid attention as they are appropriate for describing the real-world environments of reasonable evolvability and complexity [18,27]. Also, they have been used as testbed problems for comparing the performance of algorithms in the area of evolutionary computation [15,22,32,33,35,39]. The problem of maximizing k-bounded functions is NP-hard even for $k=2$ as it is at least as hard as the MAX-2-SAT problem [19]. In general, we may control the difficulty of the problem by changing the degree of dependency among the input bits (the value of k). There are good evolutionary algorithms to approximate the maximum of a k-bounded function when the dependency among the input bits is known [16,20,32,36].

Fourier transform is a formal approach to define the dependency among the input bits of functions. There have been a number of papers addressing the problem of finding the Fourier coefficients of k-bounded functions on $\{0,1\}^n$. Kargupta and Park [25] presented a deterministic algorithm using $\mathcal{O}(n^k)$ function evaluations. Later, Heckendorn and Wright [23] proposed a randomized algorithm for the problem. For the k-bounded functions with $\mathcal{O}(n)$ non-zero Fourier coefficients from a random model, they analyzed the algorithm to show that, with negligible error probability, it finds the Fourier coefficients in $\mathcal{O}(n^2\log n)$ function evaluations on average provided k is constant. For the k-bounded functions with m non-zero Fourier coefficients for constant k, Choi, Jung, and Moon [13] proved that any randomized algorithm requires $\Omega(\frac{m\log n}{\log m})$ function evaluations to find the Fourier coefficients with error probability at most a given constant if $m \leqslant n^{k-\alpha}$ for any constant $\alpha > 0$. By analyzing the algorithm of Heckendorn and Wright, they also proved that $\mathcal{O}(\lambda(n,m)m\log n)$ function evaluations are enough to find the Fourier coefficients with negligible error probability, where $\lambda(n,m)$ is between $n^{\frac{1}{2}}$ and n depending on m. Recently, for 2-bounded functions of which non-zero Fourier coefficients are between n^{-a} and n^b in absolute value for some positive constants a and b, Choi and Kim [14] showed that there exists a deterministic algorithm using $\mathcal{O}(\frac{m\log n}{\log m})$ function evaluations, provided that m is at least a polylog of n. This algorithm is non-adaptive while the previous algorithms are adaptive.[4] However, an explicit construction of the algorithm is unknown.

Our main result is

Theorem 1.1. *Suppose that f is a k-bounded function defined on $\{0,1\}^n$ for constant k and that f has m non-zero Fourier coefficients. Then, there exists an adaptive algorithm to find the Fourier coefficients of f in $\mathcal{O}(m\log n)$ function evaluations with error probability $\mathcal{O}(\frac{1}{n})$.*

We prove Theorem 1.1 by constructing a desired algorithm. Actually, we may use the same proof of the theorem to reduce the error probability to $\mathcal{O}(\frac{1}{n^a})$ for arbitrarily large constant $a > 0$. This result improves the best known upper bound $\mathcal{O}(\lambda(n,m)m\log n)$ by a factor of $\Omega(n^{\frac{1}{2}})$. In terms of learning theory, the proposed algorithm is the first *attribute-efficient* one in that its query complexity does not depend on n if we ignore the $\log n$ factor. The complexity is attractive especially when m is small compared to n, which is relevant to the fitness functions modeling the cellular dynamics mainly depending on a few genes among a large number of total genes. On the other hand, it is almost tight with respect to the lower bound $\Omega(\frac{m\log n}{\log m})$ that is shown to be valid for all m in Section 7.

We should note that there are a number of papers addressing the problem of finding the Fourier coefficients of Boolean functions [10,11,17,24,28–30]. The algorithms for Boolean functions in the literature can be extended to pseudo-Boolean functions. However, the extensions of the algorithms do not give a good bound for k-bounded functions. One of the main reasons is that their query complexities depend on the values of the target function. For example, for a k-bounded function f, the most efficient extension [17] among those has the query complexity of $\Omega(\frac{\mathrm{Var(f)}}{\theta^2})$, where $\mathrm{Var}(f)$ is the variance of f for uniform random inputs and θ is the minimum absolute value of the non-zero Fourier coefficients of f. Thus, the query complexity may be arbitrarily large as we may choose f with arbitrarily large $\mathrm{Var}(f)$ and arbitrarily small θ. The query complexity of our algorithm, on the other hand, is independent of the values of the target function.

To prove Theorem 1.1, we first show that the problem of finding the Fourier coefficients of k-bounded functions is reduced to the problem of finding k-bounded hypergraphs with a certain type of queries under a probabilistic oracle. A hypergraph is *k-bounded* if the size of each hyperedge is at most k. For a pseudo-Boolean function f on $\{0,1\}^n$, we consider the hypergraph representing the dependency among the input bits as follows. Suppose that H is a non-empty subset of $[n]$, where $[n] := \{1,\ldots,n\}$. We say that there is a *linkage* among the input bits in H if there is $H' \supseteq H$ for which the Fourier coefficient of f is non-zero.[5] This means that for any additive expression $f = \sum_i f_i$, H is included in the support set of some f_j, that is, the set of input bits on which f_j actually depends. The *linkage graph* of f is a hypergraph $G_f = ([n], E)$, where each bit in $[n]$ is a vertex and a subset H of $[n]$ belongs to the edge set E if and only if there is

[4] An algorithm is called *adaptive* if the algorithm uses a sequence of queries in which some queries depend on the previous queries. Otherwise, it is called *non-adaptive*.

[5] The term linkage means the interaction among the genes in evolutionary computation.

a linkage among the bits in H. By definition, linkage graphs have the *hierarchical property* among hyperedges that any non-empty subset of a hyperedge is also a hyperedge.

For constant k, the Fourier coefficients of a k-bounded function can be efficiently found when its linkage graph is known [23]. This fact is described in Section 2. In a hypergraph, we say that a hyperedge *crosses among* certain disjoint sets of vertices if the size of the hyperedge is equal to the number of the sets and each of the sets contains a vertex in the hyperedge. By definition, a hyperedge of size one crosses among any set of vertices including the hyperedge. Our main contribution is to show that the existence of a hyperedge crossing among a given collection of sets of vertices is efficiently testable with one-sided error.

Theorem 1.2. *Suppose that f is a k-bounded function on $\{0,1\}^n$ and $S_1,\ldots,S_j$ are j disjoint subsets of $[n]$. Then, we can use 2^j function evaluations of f to test the existence of a hyperedge in the linkage graph G_f crossing among S_i's, where the test result is correct with probability at least $\frac{1}{2^{2k}}$ if such a hyperedge exists and it is correct with probability 1 otherwise.*

This is an extension of a theorem of Heckendorn and Wright [23] (see Proposition 2.1 in Section 2), which is valid only if each S_i is a singleton set of vertices. To prove Theorem 1.2, we devise a perturbation method to test the existence of a hyperedge by flipping the bits of a random string in some or all of S_i's and aggregating the function values at the flipped strings.

Theorem 1.2 implies that the problem of finding the linkage graph of a k-bounded function is reduced to the following problem of finding hypergraphs. Suppose that a hypergraph G has n vertices and m hyperedges and the hyperedges of G are unknown. A *cross-detecting query* asks the existence of a hyperedge crossing among certain disjoint sets of vertices. We assume the *oracle with one-sided error δ* as follows. Given a cross-detecting query, the oracle correctly answers with probability at least $1-\delta$ if the true answer for the query is YES and it correctly answers with probability 1 otherwise. The problem is to find the hyperedges of G by using as few queries to the oracle as possible.

Notice that, to get Theorem 1.1, it is enough to consider only the hypergraphs with the hierarchical property. In fact, the property is a crucial observation in the algorithm of Heckendorn and Wright [23]. We consider the problem for arbitrary hypergraphs, which is of self interest, to show that the hierarchical property is not essential in obtaining our result.

Theorem 1.3. *Suppose that G is an unknown k-bounded hypergraph with n vertices and m edges for constant k. Then, for any constant $0 \leqslant \delta < 1$, the hyperedges of G can be found with error probability $\mathcal{O}(\frac{1}{n})$ by using $\mathcal{O}(m \log n)$ cross-detecting queries under the oracle with one-sided error δ.*

Theorem 1.1 is obtained from Theorems 1.2 and 1.3 and the relationship between the problems of finding Fourier coefficients and linkage graphs.

The algorithm for Theorem 1.3 is optimal up to a constant factor provided that $m \leqslant n^{k-\alpha}$ for any constant $\alpha > 0$, which follows by the lower bound shown in Section 7. It iteratively uses binary search to find the hyperedges. In this sense, it is analogous to the algorithm of Angluin and Chen [5–7] or to that of Reyzin and Srivastava [37] for some problems of finding graphs or hypergraphs. Unlike the problems, however, the answers of the oracle may contain errors in our problem and we need to handle the error bound more carefully, which is the main task in proving Theorem 1.3. A large deviation result for a sum of independent random variables with geometric distribution is crucially used for the task. There have been a number of papers addressing the problem of finding graphs or hypergraphs with various types of queries. For example, see [1–4,7–9,14,21].

The rest of the paper is organized as follows. In Section 2, we review some previous results for finding the Fourier coefficients of k-bounded functions. In Section 3, we prove Theorem 1.2, which states the linkage testability of linkage graphs, by proving relevant lemmas. Section 4 deals with the graph finding problem with cross-detecting queries under the probabilistic oracle as an independent problem. In the section, we give a randomized algorithm for the problem and analyze it to obtain Theorem 1.3. In Section 5, some remarks on the query and time complexity of the proposed algorithm are provided. In Section 6, we discuss some improvement factors in finding the Fourier coefficients of k-bounded functions including the hierarchical property. Lower bounds for finding Fourier coefficients and for finding hypergraphs with cross-detecting queries are proven in Section 7. Finally, concluding remarks close the paper in Section 8.

2. Preliminaries

2.1. Linkage test function

Munetomo and Goldberg [33] proposed a perturbation method to test the linkage in a set of two input bits. It checks the nonlinearity between the two bits by flipping the two bits of a given string individually and simultaneously and aggregating the function values at the flipped strings. Heckendorn and Wright [23] generalized this to the sets of input bits of arbitrary size. Suppose that f is a function on $\{0,1\}^n$, S is a subset of $[n]$, and x is a string in $\{0,1\}^n$. They considered the *linkage test function* $\mathfrak{L}$ depending on f, S, and x as follows

$$\mathfrak{L}(f,S,x) := \sum_{A\subseteq S} (-1)^{|A|} f(x \oplus 1_A). \tag{2.1}$$

Here, 1_A represents the string consisting of ones in the bit positions of A and zeros in the rest. For strings $x, y \in \{0,1\}^n$, $x \oplus y$ means the bitwise addition modulo 2 of x and y. The linkage test function $\mathfrak{L}$ performs a series of function evaluations at x and the strings obtained by flipping x in order to detect the linkage among the bits in S. Heckendorn and Wright [23] proved the following, which shows the usefulness of the linkage test function in finding the hyperedges of G_f.

Proposition 2.1 *(Heckendorn–Wright). Suppose that f is a k-bounded function on $\{0,1\}^n$. Then, the following holds*:

(a) *A subset S of $[n]$ is a hyperedge of G_f if and only if $\mathfrak{L}(f, S, x) \neq 0$ for some string $x \in \{0,1\}^n$.*
(b) *For a hyperedge S of size j in G_f, the probability that $\mathfrak{L}(f, S, x) \neq 0$ for a string x chosen uniformly at random from $\{0,1\}^n$ is at least $\frac{1}{2^{k-j}}$.*

By Proposition 2.1, the linkage test function determines whether a set of input bits is a hyperedge or not with one-sided error. Thus, by repeatedly evaluating the linkage test function for random strings, we can make the error arbitrarily small. In particular, when k is constant, a constant number of linkage tests (consequently, a constant number of function evaluations) is enough to determine the existence of a hyperedge with error probability at most a given constant. The algorithm of Heckendorn and Wright is based on this fact. In addition, it exploits the hierarchical property among hyperedges to reduce the number of candidate sets to be tested for hyperedges. In the algorithm, from small to large size, only the sets are tested all of which non-empty proper subsets have been shown to be hyperedges.

2.2. A Fourier transform

Walsh transform is a Fourier transform for the space of pseudo-Boolean functions in which a pseudo-Boolean function is represented as a linear combination of 2^n basis functions called *Walsh functions* [40]. For each subset H of $[n]$, the Walsh function corresponding to H, $\psi_H : \{0,1\}^n \to \mathbb{R}$, is defined as

$$\psi_H(x) := (-1)^{\sum_{i \in H} x[i]},$$

where $x[i]$ represents the ith bit value in x. If we define an inner product of two pseudo-Boolean functions f and g as

$$\langle f, g \rangle := \sum_{x \in \{0,1\}^n} \frac{f(x) \cdot g(x)}{2^n},$$

the set $\{\psi_H \mid H \subseteq [n]\}$ of Walsh functions becomes an orthonormal basis of the space of pseudo-Boolean functions. Hence, a pseudo-Boolean function f can be represented as

$$f = \sum_{H \subseteq [n]} \widehat{f}(H) \cdot \psi_H,$$

where $\widehat{f}(H) = \langle f, \psi_H \rangle$ is called the *Fourier coefficient* for H. Specifically, if $\widehat{f}(H) \neq 0$ and $\widehat{f}(H') = 0$ for any proper superset H' of H, $\widehat{f}(H)$ is called a *maximal non-zero Fourier coefficient* of f.

In [23], a number of results are provided to show the relationship between the linkage test function and the Fourier coefficients. Some of them are summarized in the following.

Proposition 2.2 *(Heckendorn–Wright). Suppose that f is a pseudo-Boolean function on $\{0,1\}^n$. Then, the following holds*:

(a) *For a subset H of $[n]$, $\widehat{f}(H)$ is a maximal non-zero Fourier coefficient of f if and only if H is a maximal hyperedge of G_f.*
(b) *For a maximal hyperedge $H \subseteq [n]$,*

$$\widehat{f}(H) = \frac{\mathfrak{L}(f, H, 0^n)}{2^{|H|}}.$$

(c) *For a subset H of $[n]$,*

$$\widehat{f}(H) = \frac{\mathfrak{L}(f, H, 0^n)}{2^{|H|}} - \sum_{H' \supsetneq H} \widehat{f}(H').$$

Once the linkage graph of f is given, we may use Proposition 2.2 to find the Fourier coefficients of f. First, Proposition 2.2(a) says that the maximal hyperedges of the linkage graph are exactly the sets of input bits with maximal non-zero Fourier coefficients. Also, for those sets H, the value $\widehat{f}(H)$ may be computed in $2^{|H|}$ function evaluations by Proposition 2.2(b) and Eq. (2.1). Then, the Fourier coefficients for the subsets of H can be found by successively applying Proposition 2.2(c). As mentioned in [23], no additional function evaluations are required to find those values. This is because if A is a proper subset of H and the value $\widehat{f}(H)$ has been obtained as above, the function evaluations of f required to get

the value $\mathfrak{L}(f, A, 0^n)$ were already performed in the course of computing $\mathfrak{L}(f, H, 0^n)$ as each subset of A is also a subset of H. Hence, if f is a k-bounded function with m non-zero Fourier coefficients and the linkage graph of f is given, at most $2^k m$ function evaluations are enough to find the Fourier coefficients of f. This implies that an upper bound for finding linkage graphs is valid as an upper bound for finding Fourier coefficients provided that k is constant and the bound for finding linkage graphs is $\Omega(m)$.

3. Generalized linkage test

3.1. Generalized linkage test function and its properties

Let f be a pseudo-Boolean function on $\{0,1\}^n$, $\mathcal{S}$ be a collection of disjoint subsets of $[n]$, and x be a string in $\{0,1\}^n$. We define the *generalized linkage test function* $\mathfrak{L}^*$ depending on f, $\mathcal{S}$, and x as follows

$$\mathfrak{L}^*(f, \mathcal{S}, x) := \sum_{\mathcal{S}' \subseteq \mathcal{S}} (-1)^{|\mathcal{S}'|} f\left(x \oplus \left(\bigoplus_{A \in \mathcal{S}'} 1_A \right)\right).$$

If we let $\mathcal{S}_H = \{\{a\} \mid a \in H\}$ for a subset H of $[n]$, we see that $\mathfrak{L}^*(f, \mathcal{S}_H, x) = \mathfrak{L}(f, H, x)$ for any $x \in \{0,1\}^n$.

The following lemmas describe the basic properties of the generalized linkage test function.

Lemma 3.1. *Suppose that $\mathcal{S}$ is a collection of disjoint subsets of $[n]$. Then, the following holds*:

(a) (*Linearity*) *If $f_1, \ldots, f_\ell$ are pseudo-Boolean functions on $\{0,1\}^n$ and $c_1, \ldots, c_\ell$ are constants,*

$$\mathfrak{L}^*\left(\sum_{i=1}^{\ell} c_i f_i, \mathcal{S}, x \right) = \sum_{i=1}^{\ell} c_i \mathfrak{L}^*(f_i, \mathcal{S}, x)$$

for all $x \in \{0,1\}^n$.

(b) (*Recursion*) *If f is a pseudo-Boolean function on $\{0,1\}^n$,*

$$\mathfrak{L}^*(f, \mathcal{S}, x) = \mathfrak{L}^*\big(f, \mathcal{S} \setminus \{A\}, x\big) - \mathfrak{L}^*\big(f, \mathcal{S} \setminus \{A\}, x \oplus 1_A\big)$$

for any $A \in \mathcal{S}$ and any $x \in \{0,1\}^n$.

Proof. Lemma 3.1(a) follows as $\mathfrak{L}^*(c_i f_i, \mathcal{S}, x) = c_i \mathfrak{L}^*(f_i, \mathcal{S}, x)$ and $\mathfrak{L}^*(f_i + f_j, \mathcal{S}, x) = \mathfrak{L}^*(f_i, \mathcal{S}, x) + \mathfrak{L}^*(f_j, \mathcal{S}, x)$ for all $x \in \{0,1\}^n$. For any $A \in \mathcal{S}$, Lemma 3.1(b) follows because

$$\begin{aligned}
\mathfrak{L}^*(f, \mathcal{S}, x) &= \sum_{\mathcal{S}' \subseteq \mathcal{S}} (-1)^{|\mathcal{S}'|} f\big(x \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) \\
&= \sum_{\mathcal{S}' \subseteq \mathcal{S}:\ A \notin \mathcal{S}'} (-1)^{|\mathcal{S}'|} f\big(x \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) + \sum_{\mathcal{S}' \subseteq \mathcal{S}:\ A \in \mathcal{S}'} (-1)^{|\mathcal{S}'|} f\big(x \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) \\
&= \sum_{\mathcal{S}' \subseteq \mathcal{S}:\ A \notin \mathcal{S}'} (-1)^{|\mathcal{S}'|} f\big(x \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) - \sum_{\mathcal{S}' \subseteq \mathcal{S}:\ A \notin \mathcal{S}'} (-1)^{|\mathcal{S}'|} f\big(x \oplus 1_A \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) \\
&= \sum_{\mathcal{S}' \subseteq \mathcal{S} \setminus \{A\}} (-1)^{|\mathcal{S}'|} f\big(x \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) - \sum_{\mathcal{S}' \subseteq \mathcal{S} \setminus \{A\}} (-1)^{|\mathcal{S}'|} f\big(x \oplus 1_A \oplus (\oplus_{A' \in \mathcal{S}'} 1_{A'})\big) \\
&= \mathfrak{L}^*\big(f, \mathcal{S} \setminus \{A\}, x\big) - \mathfrak{L}^*\big(f, \mathcal{S} \setminus \{A\}, x \oplus 1_A\big)
\end{aligned}$$

for all $x \in \{0,1\}^n$. □

Lemma 3.2. *Suppose that f is a pseudo-Boolean function on $\{0,1\}^n$ and $\mathcal{S}$ is a collection of disjoint subsets of $[n]$. If the support set of f is disjoint with some $A \in \mathcal{S}$, then $\mathfrak{L}^*(f, \mathcal{S}, x) = 0$ for all $x \in \{0,1\}^n$.*

Proof. By Lemma 3.1(b), for any $x \in \{0,1\}^n$

$$\mathfrak{L}^*(f, \mathcal{S}, x) = \mathfrak{L}^*\big(f, \mathcal{S} \setminus \{A\}, x\big) - \mathfrak{L}^*\big(f, \mathcal{S} \setminus \{A\}, x \oplus 1_A\big).$$

Since the support set of f is disjoint with A, $f(x) = f(x \oplus 1_A)$ and so $\mathfrak{L}^*(f, \mathcal{S} \setminus \{A\}, x \oplus 1_A) = \mathfrak{L}^*(f, \mathcal{S} \setminus \{A\}, x)$. This completes the proof. □

3.2. Linkage test theorem

In this section, we prove the main theorem for randomized linkage test to obtain Theorem 1.2. Before stating the theorem, we introduce some terminology and notation. Suppose that $\mathcal{S} = \{S_1, \ldots, S_j\}$ is a collection of disjoint subsets of $[n]$. A collection $\mathcal{R} = \{R_1, \ldots, R_j\}$ of subsets of $[n]$ is called a *setwise subcollection* of $\mathcal{S}$ if $R_i \subseteq S_i$ for all $i = 1, \ldots, j$. In this case, we denote by $\mathcal{R} \Subset \mathcal{S}$. Note that a setwise subcollection $\mathcal{R}$ of $\mathcal{S}$ may contain multiple empty sets. We consider a random model $\Gamma(\mathcal{S})$ that generates a setwise subcollection of $\mathcal{S}$ as follows: For each i, select each element in S_i independently and with probability $\frac{1}{2}$ and let R_i be the set of selected elements. Then, we have a setwise subcollection $\mathcal{R} = \{R_i \mid i = 1, \ldots, j\}$ of $\mathcal{S}$. In the following, $\mathsf{str}(\mathcal{R})$ denotes the set of strings x such that for each i, the values of x are the same in the bit positions belonging to R_i:

$$\mathsf{str}(\mathcal{R}) := \left\{x \in \{0,1\}^n \;\middle|\; x[a] = x[b] \text{ if } a \text{ and } b \text{ belong to some } R_i\right\}.$$

Theorem 3.3. *Suppose that f is a k-bounded function on $\{0,1\}^n$ and $\mathcal{S}$ is a collection of disjoint subsets of $[n]$. Then, the following holds*:

(a) *The linkage graph G_f has a hyperedge crossing among $\mathcal{S}$ if and only if there exist $\mathcal{R} \Subset \mathcal{S}$ and $x \in \mathsf{str}(\mathcal{R})$ such that $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$.*
(b) *If G_f has a hyperedge crossing among $\mathcal{S}$, the probability that $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$ is at least $\frac{1}{2^{2k}}$ for $\mathcal{R}$ randomly generated from $\Gamma(\mathcal{S})$ and x chosen uniformly at random from $\mathsf{str}(\mathcal{R})$.*

Theorem 3.3 implies Theorem 1.2 and provides an efficient method to test for the existence of a hyperedge crossing among a given collection of sets of vertices in the linkage graph.

To prove the main theorem, we consider a random function obtained from f and $\mathcal{R}$ that is useful to describe the behavior of $\mathfrak{L}^*(f, \mathcal{R}, x)$ over x in $\mathsf{str}(\mathcal{R})$. We first present the definition of the function. Suppose that the size of $\mathcal{S}$ is j and $\mathcal{R} = \{R_1, \ldots, R_j\}$ is a setwise subcollection of $\mathcal{S}$ with R_i being non-empty for all i. Letting $\rho = \sum_{i=1}^{j} |R_i|$, we define $f_{\mathcal{R}} : \{0,1\}^{n-\rho+j} \to \mathbb{R}$ as follows. Among the $n - \rho + j$ bit positions that $f_{\mathcal{R}}$ depends on, j positions represent R_i, $i = 1, \ldots, j$ and $n - \rho$ positions correspond to the bit positions in $\bar{R} := [n] \setminus (\bigcup_i R_i)$. More precisely, for each R_i set a distinct bit position a_i with $1 \leqslant a_i \leqslant j$ and for each $i \in \bar{R}$ set a distinct bit position b_i with $j + 1 \leqslant b_i \leqslant n - \rho + j$. The function $f_{\mathcal{R}}$ is a sum of the subfunctions $\varphi_{H,\mathcal{R}}$ over $H \subseteq [n]$:

$$f_{\mathcal{R}} := \sum_{H \subseteq [n]} \widehat{f}(H)\varphi_{H,\mathcal{R}}.$$

Here, for $H \subseteq [n]$, $\varphi_{H,\mathcal{R}} : \{0,1\}^{n-\rho+j} \to \mathbb{R}$ is defined as, for all $y \in \{0,1\}^{n-\rho+j}$

$$\varphi_{H,\mathcal{R}}(y) := \begin{cases} (-1)^{\sum_{i=1}^{j} y[a_i] + \sum_{i \in H \cap \bar{R}} y[b_i]} & \text{if the size of } H \cap R_i \text{ is odd for all } i, \\ 0 & \text{otherwise.} \end{cases}$$

Now, we present two lemmas for $f_{\mathcal{R}}$. To this end, we give a natural one-to-one and onto mapping from $\mathsf{str}(\mathcal{R})$ to the domain of $f_{\mathcal{R}}$. For each string x in $\mathsf{str}(\mathcal{R})$, let $x_{\mathcal{R}}$ be the string in $\{0,1\}^{n-\rho+j}$ such that $x_{\mathcal{R}}[a_i] = x[r_i]$ for some $r_i \in R_i$ for all $i = 1, \ldots, j$ and $x_{\mathcal{R}}[b_i] = x[i]$ for all $i \in \bar{R}$. Also, let $S_{\mathcal{R}} := \{a_i \mid i = 1, \ldots, j\}$ which is a set of bit positions in the domain of $f_{\mathcal{R}}$. The following shows that $\mathfrak{L}^*$ of f and $\mathcal{R}$ behaves the same as $\mathfrak{L}$ of $f_{\mathcal{R}}$ and $S_{\mathcal{R}}$ under some non-empty condition on $\mathcal{R}$.

Lemma 3.4. *Suppose that R_i is non-empty for all $i = 1, \ldots, j$. Then,*

$$\mathfrak{L}^*(f, \mathcal{R}, x) = \mathfrak{L}(f_{\mathcal{R}}, S_{\mathcal{R}}, x_{\mathcal{R}})$$

for all $x \in \mathsf{str}(\mathcal{R})$.

Proof. By linearity of $\mathfrak{L}^*$ and $\mathfrak{L}$, it is enough to show that for all $x \in \mathsf{str}(\mathcal{R})$

$$\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = \mathfrak{L}(\varphi_{H,\mathcal{R}}, S_{\mathcal{R}}, x_{\mathcal{R}}).$$

First, suppose that the size of $H \cap R_i$ is odd for all i. Then, it holds that for all $x \in \mathsf{str}(\mathcal{R})$, $\psi_H(x) = \varphi_{H,\mathcal{R}}(x_{\mathcal{R}})$ since for each i, the sum of bit values of x in $H \cap R_i$ is $x_{\mathcal{R}}[a_i]$ modulo 2 and so

$$\psi_H(x) = (-1)^{\sum_{i=1}^{j} \sum_{r \in H \cap R_i} x[r] + \sum_{i \in H \cap \bar{R}} x[i]} = (-1)^{\sum_{i=1}^{j} x_{\mathcal{R}}[a_i] + \sum_{i \in H \cap \bar{R}} x_{\mathcal{R}}[b_i]} = \varphi_{H,\mathcal{R}}(x_{\mathcal{R}}).$$

If we let $1_{\mathcal{R}'} := \bigoplus_{R_i \in \mathcal{R}'} 1_{R_i}$ for $\mathcal{R}' \subseteq \mathcal{R}$,

$$\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = \sum_{\mathcal{R}' \subseteq \mathcal{R}} (-1)^{|\mathcal{R}'|} \psi_H(x \oplus 1_{\mathcal{R}'}).$$

Notice that if x is in $\mathrm{str}(\mathcal{R})$, $x \oplus 1_{\mathcal{R}'}$ is also in $\mathrm{str}(\mathcal{R})$ and it is mapped to $x_{\mathcal{R}} \oplus 1_{S_{\mathcal{R}'}}$ in $\{0,1\}^{n-\rho+j}$ for $S_{\mathcal{R}'} := \{a_i \mid R_i \in \mathcal{R}'\}$. Hence,

$$\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = \sum_{\mathcal{R}' \subseteq \mathcal{R}} (-1)^{|\mathcal{R}'|} \psi_H(x \oplus 1_{\mathcal{R}'}) = \sum_{S_{\mathcal{R}'}: \mathcal{R}' \subseteq \mathcal{R}} (-1)^{|S_{\mathcal{R}'}|} \varphi_{H,\mathcal{R}}(x_{\mathcal{R}} \oplus 1_{S_{\mathcal{R}'}}) = \mathfrak{L}(\varphi_{H,\mathcal{R}}, S_{\mathcal{R}}, x_{\mathcal{R}}).$$

Now, suppose that the size of $H \cap R_i$ is even for some i. Then, for all $x \in \{0,1\}^n$, $\psi_H(x) = \psi_H(x \oplus 1_{R_i})$ and so $\mathfrak{L}^*(\psi_H, \mathcal{R} \setminus \{R_i\}, x) = \mathfrak{L}^*(\psi_H, \mathcal{R} \setminus \{R_i\}, x \oplus 1_{R_i})$. Thus, $\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = 0$ by Lemma 3.1(b). The desired equality follows since by definition, $\varphi_{H,\mathcal{R}}$ is the zero function and so $\mathfrak{L}(\varphi_{H,\mathcal{R}}, S_{\mathcal{R}}, y) = 0$ for all $y \in \{0,1\}^{n-\rho+j}$. □

Next, we show that for $\mathcal{R}$ generated from $\Gamma(\mathcal{S})$, the probability of $G_{f_{\mathcal{R}}}$ having the hyperedge $S_{\mathcal{R}}$ is at least a constant if G_f has a hyperedge crossing among $S_{\mathcal{R}}$. To prove it, we need the following for multilinear polynomials of Boolean random variables. It is obtained from the paper of Schwartz [38]. (See also [34].)

Proposition 3.5 *(Schwartz). Suppose that $t_1, \ldots, t_n$ are independent random variables such that t_i is 0 or 1 each with probability $\frac{1}{2}$ for all $i = 1, \ldots, n$. If $g(t_1, \ldots, t_n)$ is a non-zero multilinear polynomial of degree j or less in $t_1, \ldots, t_n$, then the probability of $g(t_1, \ldots, t_n) \neq 0$ is at least $\frac{1}{2^j}$.*

Lemma 3.6. *Suppose that G_f has a hyperedge crossing among $\mathcal{S}$. For $\mathcal{R} = \{R_1, \ldots, R_j\}$ randomly generated from $\Gamma(\mathcal{S})$, the probability that R_i is non-empty for all $i = 1, \ldots, j$ and $G_{f_{\mathcal{R}}}$ has the hyperedge $S_{\mathcal{R}}$ is at least $\frac{1}{2^{k+j}}$.*

Proof. Let $\mathcal{S} = \{S_1, \ldots, S_j\}$ with $R_i \subseteq S_i$ for $i = 1, \ldots, j$. Since G_f has a hyperedge crossing among $\mathcal{S}$, there exist one or more sets $H \subseteq [n]$ such that $\widehat{f}(H) \neq 0$ and $H \cap S_i \neq \emptyset$ for all i. Among those, we choose a maximal set H^* in the size of the intersection with S_i, incrementally for i from 1 to j. More precisely, for i from 1 to j, $|H^* \cap S_i| \geqslant |H \cap S_i|$ over $H \subseteq [n]$ such that $\widehat{f}(H) \neq 0$, $H \cap S_{i'} \neq \emptyset$ for all i', and $|H \cap S_{i'}| = |H^* \cap S_{i'}|$ for $i' = 1, \ldots, i-1$. For each i choose an element in $H^* \cap S_i$ and let A_i be the singleton set of the element. Let B_i be the set of the other elements in $H^* \cap S_i$. Since S_i's are disjoint and the size of H^* is at most k as f is k-bounded, $\sum_i |A_i \cup B_i| = \sum_i |H^* \cap S_i| \leqslant |H^*| \leqslant k$. Thus, the probability that $R_i \supseteq A_i$ and $R_i \cap B_i = \emptyset$ for all i is at least $\frac{1}{2^k}$.

Let $\mathcal{R}$ be conditioned on $R_i \cap (A_i \cup B_i)$ such that $R_i \supseteq A_i$ and $R_i \cap B_i = \emptyset$ for all i. Letting $A := \bigcup_i A_i$, we consider the Fourier coefficient $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}})$ for $H^*_{\mathcal{R}} \subseteq [n - \rho + j]$ with

$$H^*_{\mathcal{R}} := \{a_i \mid i = 1, \ldots, j\} \cup \left\{ b_i \;\middle|\; i \in H^* \setminus A \right\}.$$

We will show that the conditional probability of $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}}) \neq 0$ is at least $\frac{1}{2^j}$. If $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}}) \neq 0$, this means that $H^*_{\mathcal{R}}$ and its non-empty subsets are hyperedges of $G_{f_{\mathcal{R}}}$. Since $S_{\mathcal{R}}$ is a subset of $H^*_{\mathcal{R}}$, the lemma follows.

Let $S := \bigcup_i S_i$. For each $i \in S \setminus H^*$, denote by t_i the random variable such that $t_i = 1$ if i is chosen for $\mathcal{R}$ and $t_i = 0$ otherwise. Denote

$$\mathcal{H}^* := \left\{ H \subseteq [n] \;\middle|\; \widehat{f}(H) \neq 0,\ H \setminus \left(\bigcup_i (S_i \setminus B_i) \right) = H^* \setminus A,\ \text{and } \left| H \cap (S_i \setminus B_i) \right| = 1 \text{ for all } i \right\}.$$

It follows that

$$\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}}) = \sum_{H \in \mathcal{H}^*} \widehat{f}(H) \prod_{i \in H \setminus H^*} t_i. \tag{3.1}$$

To see this, note that $\varphi_{H^*,\mathcal{R}}(x) = \psi_{H^*_{\mathcal{R}}}(x_{\mathcal{R}})$ for all x in $\mathrm{str}(\mathcal{R})$ and so $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}}) = \sum_H \widehat{f}(H)$, where the summation is over $H \subseteq [n]$ such that $\varphi_{H,\mathcal{R}} = \varphi_{H^*,\mathcal{R}}$. By definition, $\varphi_{H,\mathcal{R}} = \varphi_{H^*,\mathcal{R}}$ if and only if $H \setminus (\bigcup_i R_i) = H^* \setminus (\bigcup_i R_i)$ and the size of $H \cap R_i$ is odd for all i as the size of $H^* \cap R_i = A_i$ is 1 for all i. This means that $H \setminus (\bigcup_i (S_i \setminus B_i)) = H^* \setminus A$ and the size of $(H \cap (S_i \setminus B_i)) \cap R_i$ is odd for all i as $R_i \supseteq A_i$ and $R_i \cap B_i = \emptyset$ for all i. Since H^* was chosen in the maximal sense as above, the size of $H \cap (S_i \setminus B_i)$ is at most 1 for all i if $H \setminus (\bigcup_i (S_i \setminus B_i)) = H^* \setminus A$. Therefore, $\widehat{f}(H)$ contributes to $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}})$ if and only if $H \setminus (\bigcup_i (S_i \setminus B_i)) = H^* \setminus A$, the size of $H \cap (S_i \setminus B_i)$ is one for all i, and the element in $H \cap (S_i \setminus B_i) \setminus A_i$ if it exists is chosen for $\mathcal{R}$ for all i. Since $\bigcup_i (H \cap (S_i \setminus B_i) \setminus A_i) = H \setminus H^*$, we obtain Eq. (3.1).

Then, $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}})$ is a non-zero polynomial in t_i's as H^* belongs to $\mathcal{H}^*$ and so $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}})$ has the non-zero constant term $\widehat{f}(H^*)$. Also, it is of degree j or less as the size of $H \setminus H^*$ for $H \in \mathcal{H}$ is at most j. Since t_i's are independent, the conditional probability that $\widehat{f_{\mathcal{R}}}(H^*_{\mathcal{R}}) \neq 0$ is at least $\frac{1}{2^j}$ by Proposition 3.5 as desired. □

Now, we prove Theorem 3.3.

```
GraphFindingAlgorithm(n,k,δ)
// E_j : the set of the hyperedges of size j found so far
// Q : the set of the vertices in the hyperedges of size j found so far
// W : the set of the vertices v such that all the hyperedges of size j containing v have
been found by the algorithm
    for j from 1 to k
        Q ← ∅, W ← ∅, E_j ← ∅;
        repeat
            (S_i)_{i=1}^{j} ← CheckExistence(∅,W,j);
            if (S_i)_{i=1}^{j} = NULL, break;
            v ← BinarySearch((S_i)_{i=1}^{j},1);
            Q ← Q ∪ {v};
            while Q \ W ≠ ∅
                choose a vertex v in Q \ W;
                E_{v,j} ← FindHyperedges({v},W,j);
                E_j ← E_j ∪ E_{v,j};
                Q ← Q ∪ (⋃_{H∈E_{v,j}} H);
                W ← W ∪ {v};
    E ← ⋃_{j=1}^{k} E_j;
    return E;
```

Fig. 1. Main procedure of GFA.

Proof of Theorem 3.3. Since (b) implies the only-if part of (a), it is enough to prove (b) and then prove the if part of (a).

Suppose that G_f has a hyperedge crossing among $\mathcal{S}$. Then, by Lemma 3.6, the event occurs with probability $\frac{1}{2^{k+j}}$ or more that for $\mathcal{R} = \{R_1, \ldots, R_j\}$ randomly generated from $\Gamma(\mathcal{S})$, R_i is non-empty for all i and $G_{f_\mathcal{R}}$ has the hyperedge $S_\mathcal{R}$. Under the condition of the event, the probability that $\mathfrak{L}(f_\mathcal{R}, S_\mathcal{R}, y) \neq 0$ for uniform random y from $\{0,1\}^{n-\rho+j}$ is at least $\frac{1}{2^{k-j}}$ by Proposition 2.1(b) as the size of $S_\mathcal{R}$ is j and by definition, $f_\mathcal{R}$ is k-bounded. Hence, under the same condition, the probability that $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$ is at least $\frac{1}{2^{k-j}}$ for uniform random x from $\mathrm{str}(\mathcal{R})$ by Lemma 3.4 and this completes the proof of (b).

Now, we prove the if part of (a). Suppose that there exist $\mathcal{R} \Subset \mathcal{S}$ and $x \in \mathrm{str}(\mathcal{R})$ such that $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$. By Lemma 3.1(a),

$$\mathfrak{L}^*(f, \mathcal{R}, x) = \sum_{H \subseteq [n]} \widehat{f}(H) \mathfrak{L}^*(\psi_H, \mathcal{R}, x).$$

Here, by Lemma 3.2, $\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = 0$ for any $H \subseteq [n]$ such that $H \cap A = \emptyset$ for some $A \in \mathcal{R}$. Hence, among the sets H such that $H \cap A \neq \emptyset$ for all $A \in \mathcal{R}$, there is H such that $\widehat{f}(H) \neq 0$ since $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$. We may choose a subset H' of H crossing among $\mathcal{R}$. Since $\widehat{f}(H) \neq 0$, by definition, H' is a hyperedge of G_f as desired. □

4. Finding graphs with cross-detecting queries

In this section, we focus on the problem of finding hypergraphs with cross-detecting queries under the oracle with one-sided error. Section 4.1 presents a randomized algorithm for the problem. The algorithm is analyzed in Section 4.2, which induces Theorem 1.3.

4.1. Algorithm for finding graphs

In this section, we present the *Graph Finding Algorithm* (GFA) to find hypergraphs with cross-detecting queries under the oracle with one-sided error. GFA takes three arguments: The number n of the vertices of the unknown hypergraph, the size bound k of the hyperedges, and the error bound $0 \leqslant \delta < 1$ for the answer of the oracle. It returns the set of the hyperedges of the hypergraph that it has found. GFA consists of the main procedure GraphFindingAlgorithm (Fig. 1) and the three subprocedures CheckExistence (Fig. 2), BinarySearch (Fig. 3), and FindHyperedges (Fig. 4). In the pseudocode, the values of n, k, and δ can be accessed by any procedure. All other variables are local to the given procedure.

Suppose that G is an unknown hypergraph given to GFA and let G_j be the induced subgraph of G consisting of the hyperedges of size j for $1 \leqslant j \leqslant k$. GFA successively finds the hyperedges of $G_1, \ldots, G_k$. To find the hyperedges of G_j for $j = 1, \ldots, k$, it first calls CheckExistence to check whether there is a hyperedge of size j that has not been found. If such a hyperedge H exists, it finds all the hyperedges of size j in the connected component that H belongs to. To this end, it first chooses a vertex v in H by BinarySearch. Then, starting with v, it iteratively chooses a vertex in the connected component and finds all the hyperedges containing the vertex by FindHyperedges. GFA continues the above process until there is no more hyperedge that can be found.

Now, we describe each of the subprocedures in more detail. Given sets of vertices U and W and a positive integer j, CheckExistence performs a randomized test for whether there is a hyperedge of size j including U and not

```
CheckExistence(U,W,j)
    label the vertices in U as v_1, ..., v_|U|;
    for i from 1 to |U|
        S_i ← {v_i};
    repeat ⌈ (e^j √(j+1)) / (1−δ) log n ⌉ times
        for i from |U|+1 to j
            S_i ← ∅;
        for each v ∈ [n] \ (U ∪ W)
            choose i uniformly at random from {|U|+1, ..., j};
            S_i ← S_i ∪ {v};
        if CDQ(S_1, ..., S_j) = YES
            return (S_i)_{i=1}^j;
    return NULL;
```

Fig. 2. Procedure to check the existence of a hyperedge of size j including U and not intersecting with W.

```
BinarySearch((S_i)_{i=1}^j, r)
    if |S_r| = 1, return the vertex in S_r;
    repeat ⌈ 6(j+1)/(1−δ) log n ⌉ times
        choose a subset S'_r of S_r uniformly at random among the subsets of size ⌊ |S_r|/2 ⌋;
        if CDQ(S_1, ..., S_{r−1}, S'_r, S_{r+1}, ..., S_j) = YES,
            S_r ← S'_r;
            if |S_r| = 1, return the vertex in S_r;
    return a vertex in S_r;
```

Fig. 3. Procedure to search a vertex in S_r that is contained in a hyperedge of size j crossing among $S_1, \ldots, S_j$.

```
FindHyperedges(U,W,j)
    if |U| = j, return {U};
    E_{U,j} ← ∅, A ← ∅;
    repeat
        (S_i)_{i=1}^j ← CheckExistence(U, W ∪ A, j);
        if (S_i)_{i=1}^j = NULL, break;
        v ← BinarySearch((S_i)_{i=1}^j, |U|+1);
        E_{U,j} ← E_{U,j} ∪ FindHyperedges(U ∪ {v}, W ∪ A, j);
        A ← A ∪ {v};
    return E_{U,j};
```

Fig. 4. Procedure to find the hyperedges of size j including U and not intersecting with W.

intersecting with W. For this purpose, it iteratively generates random cross-detecting queries $(S_i)_{i=1}^j$ as follows. Letting $U = \{v_1, \ldots, v_{|U|}\}$, S_i is fixed with $S_i = \{v_i\}$ for $1 \leqslant i \leqslant |U|$. The sets $S_{|U|+1}, \ldots, S_j$ are generated as a uniform random partition of $[n] \setminus (U \cup W)$. If the oracle answers YES for some $(S_i)_{i=1}^j$, there is a hyperedge of size j crossing among S_i's that includes U and does not intersect with W. In this case, CheckExistence returns the sets $(S_i)_{i=1}^j$. If the oracle answers NO for all the queries, CheckExistence returns NULL indicating that there is no such hyperedge. (In Figs. 2 and 3, CDQ represents the answer of the oracle for the given cross-detecting query.)

Next, given disjoint sets of vertices $(S_i)_{i=1}^j$ and a positive integer r between 1 and j, BinarySearch returns a vertex both in S_r and in one of the hyperedges crossing among S_i's. Among the subsets of S_r of size $\lfloor \frac{|S_r|}{2} \rfloor$, it chooses a subset S'_r uniformly at random and asks the cross-detecting query for the sets of vertices $(S_i)_{i=1}^j$ in which S_r is replaced with S'_r. If the answer of the oracle is YES, i.e., if it turns out that there is a hyperedge crossing among the sets, it replaces S_r with S'_r. BinarySearch repeats this process at most a specified number of times until there remains one vertex in S_r. If there remains one vertex in S_r before the specified number of iterations, BinarySearch returns the vertex. Otherwise, it fails an exact search for the desired vertex and returns an arbitrary vertex in S_r.

Finally, given two sets of vertices U and W and a positive integer j, FindHyperedges returns the set of the hyperedges of size j that include U and do not intersect with W. In FindHyperedges, A contains the vertices such that all the desired hyperedges of size j containing the vertices have been found. Initially, A is set to be empty. If $|U| = j$, U is the only desired hyperedge and FindHyperedges returns the set consisting of U. Otherwise, it recursively finds the desired hyperedges of size j as follows. First, it calls CheckExistence to check whether there is a hyperedge of size j that includes U and does not intersect with $W \cup A$. If CheckExistence returns NULL, FindHyperedges regards that as evidence that there is no such a hyperedge and returns the set of the hyperedges found so far. Otherwise, it chooses a vertex v in the hyperedge by using BinarySearch. Then, it recursively finds all the hyperedges of size j that include $U \cup \{v\}$ and do not intersect with $W \cup A$. After that, it puts v into A and repeats the above process.

4.2. Algorithm analysis

In this section, we analyze GFA to obtain Theorem 1.3. We first bound the number of cross-detecting queries used in GFA.

Lemma 4.1. *Suppose that G is an unknown k-bounded hypergraph with n vertices and m hyperedges for constant k. Then, for any constant $0 \leqslant \delta < 1$, GFA uses $\mathcal{O}(m \log n)$ cross-detecting queries for G under the oracle with one-sided error δ.*

Proof. We show that the number of cross-detecting queries used for finding G_j is $\mathcal{O}(m \log n)$ for each $j = 1, \ldots, k$.

Since cross-detecting queries are used only in CheckExistence and BinarySearch, we count how many times the two procedures are called in GFA. Notice that in the main procedure and FindHyperedges, each call of BinarySearch is followed by a call of FindHyperedges. Thus, the number of BinarySearch calls is at most that of FindHyperedges. On the other hand, a call of CheckExistence is executed just before the repeat loop is terminated in the main procedure and FindHyperedges. Except for those calls, each call of CheckExistence is followed by a call of FindHyperedges. Thus, the number of CheckExistence calls for finding G_j is at most one plus two times that of FindHyperedges.

Now, we show that the number of FindHyperedges calls for finding G_j is at most $j^2|E(G_j)|$. Let v be a vertex chosen from $Q \setminus W$ in the while loop of the main procedure and let $\deg_j(v)$ be the degree of v in G_j, i.e., the number of hyperedges of size j containing v. The number of FindHyperedges calls in finding the hyperedges of G_j containing v is at most $j \times \deg_j(v)$. Each vertex chosen from $Q \setminus W$ in the while loop is contained in a hyperedge of G_j. Thus, the while loop iterates only for the vertices of degree one or more in G_j. Hence, the overall number of FindHyperedges calls is at most

$$\sum_{\deg_j(v)>0} j \times \deg_j(v) = j \sum_{\deg_j(v)>0} \deg_j(v) = j^2 \big|E(G_j)\big|.$$

Thus, for finding G_j, CheckExistence and BinarySearch are called at most $2j^2|E(G_j)| + 1$ and $j^2|E(G_j)|$ times, respectively. Since at most $\lceil \frac{e^j\sqrt{j+1}}{1-\delta} \log n \rceil$ and $\lceil \frac{6(j+1)}{1-\delta} \log n \rceil$ cross-detecting queries are used in the two procedures, respectively, and $|E(G_j)| \leqslant m$, the number of cross-detecting queries used for finding G_j is at most

$$\left\lceil \frac{e^j\sqrt{j+1}}{1-\delta} \log n \right\rceil \times \left(2j^2 m + 1\right) + \left\lceil \frac{6(j+1)}{1-\delta} \log n \right\rceil \times j^2 m,$$

which is $\mathcal{O}(m \log n)$ since δ is constant and $j \leqslant k$ for constant k. □

To analyze the error probability of GFA, we need a large deviation result for a sum of independent random variables following geometric distributions. A random variable X follows the geometric distribution with parameter p if, for a coin of which HEAD appears with probability p, X is the number of coin tosses until the first HEAD appears. The result for geometric distributions is easily obtained by Chernoff bound [12,31].

Proposition 4.2 *(Chernoff). Suppose that, for some $0 < p \leqslant 1$, $X_1, \ldots, X_\ell$ are independent random variables such that $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for all $1 \leqslant i \leqslant \ell$. Let $X = \sum_{i=1}^{\ell} X_i$. Then, for any $0 \leqslant \alpha < 1$,*

$$\Pr\big[X \leqslant (1-\alpha)\mathrm{E}[X]\big] \leqslant \exp\left(-\frac{\mathrm{E}[X]\alpha^2}{2}\right).$$

Lemma 4.3. *Suppose that, for some $0 < p \leqslant 1$, $X_1, \ldots, X_\ell$ are independent random variables each of which follows the geometric distribution with parameter p. Let $X = \sum_{i=1}^{\ell} X_i$. Then, for any $\alpha > 0$,*

$$\Pr\big[X > (1+\alpha)\mathrm{E}[X]\big] \leqslant \exp\left(-\frac{\alpha^2\ell}{2(1+\alpha)}\right).$$

Proof. For a coin of which HEAD appears with probability p, consider an experiment in which the coin is flipped independently and infinitely many times. Then, we may regard X as the number of coin flips until the ℓth HEAD appears. Let $r = \lceil (1+\alpha)\mathrm{E}[X] \rceil = \lceil (1+\alpha)\frac{\ell}{p} \rceil$ and let $Y_1, \ldots, Y_r$ be independent random variables such that $\Pr[Y_i = 1] = p$ and $\Pr[Y_i = 0] = 1 - p$ for all $1 \leqslant i \leqslant r$. If we let $Y = \sum_{i=1}^{r} Y_i$, then

$$\Pr\big[X > (1+\alpha)\mathrm{E}[X]\big] \leqslant \Pr[X \geqslant r] \leqslant \Pr[Y \leqslant \ell].$$

We may write $\ell = (1-\beta)\mathrm{E}[Y]$ for some fixed β with $0 \leqslant \beta < 1$ as $\mathrm{E}[Y] = rp$ and $\ell < (1+\alpha)\ell \leqslant rp$. Then, Proposition 4.2 implies

$$\Pr[Y \leqslant \ell] = \Pr\big[Y \leqslant (1-\beta)\mathrm{E}[Y]\big] \leqslant \exp\left(-\frac{\mathrm{E}[Y]\beta^2}{2}\right) = \exp\left(-\frac{rp\beta^2}{2}\right).$$

Letting $r = (1+\alpha)\frac{\ell}{p} + \gamma$ for $0 \leqslant \gamma < 1$ and substituting $\beta = \frac{rp-\ell}{rp}$, it is not difficult to show that

$$\frac{rp\beta^2}{2} \geqslant \frac{\alpha^2 \ell}{2(1+\alpha)}$$

for any $0 \leqslant \gamma < 1$ and the desired result follows. □

Lemma 4.4. *Suppose that G is an unknown k-bounded hypergraph with n vertices and m hyperedges for constant k. Then, for any $0 \leqslant \delta < 1$, GFA correctly finds the hyperedges of G with probability $1 - \mathcal{O}(\frac{1}{n})$ under the oracle with one-sided error δ.*

Proof. We show that the probability that GFA does not correctly find G_j is $\mathcal{O}(\frac{1}{n})$ for each $j = 1, \ldots, k$. Then, the lemma follows by the union bound.

For each j, it is enough to show that the probability of CheckExistence or BinarySearch performing incorrectly once or more times in the process of finding G_j is $\mathcal{O}(\frac{1}{n})$. Provided that the two procedures correctly performs in the process, it can be easily shown that for each given vertex, FindHyperedges in the main procedure correctly returns the hyperedges of G_j containing the vertex, i.e., GFA correctly finds G_j.

We first consider the probability that CheckExistence performs incorrectly for given arguments U, W, and j. If there is no hyperedge of G_j including U and not intersecting with W, CheckExistence returns NULL and the probability of CheckExistence being incorrect is zero. Suppose that there is such a hyperedge. Let $U = \{v_1, \ldots, v_{|U|}\}$ and let the hyperedge be $\{v_1, \ldots, v_{|U|}, v_{|U|+1}, \ldots, v_j\}$. The probability of the event that $v_{|U|+1}, \ldots, v_j$ are put into different S_i's is $\frac{(j-|U|)!}{(j-|U|)^{j-|U|}}$. When the event occurs, the probability that the oracle answers YES for the cross-detecting query $(S_i)_{i=1}^{j}$ is at least $1-\delta$. Hence, the probability that the hyperedge is not detected for $\lceil \frac{e^j\sqrt{j+1}}{1-\delta} \log n \rceil$ iterations of the repeat loop in CheckExistence is at most

$$\left(1 - \frac{(j-|U|)!}{(j-|U|)^{j-|U|}}(1-\delta)\right)^{\frac{e^j\sqrt{j+1}}{1-\delta}\log n}.$$

By using the fact that $1 - x \leqslant e^{-x}$ for any real x, this value is at most

$$\exp\left(-\frac{(j-|U|)!e^j\sqrt{j+1}}{(j-|U|)^{j-|U|}}\log n\right).$$

Some calculation using the facts that $\frac{(j-|U|)!}{(j-|U|)^{j-|U|}} \geqslant \frac{j!}{j^j}$ and $j! > \sqrt{2\pi j}(\frac{j}{e})^j e^{\frac{1}{12j+1}}$ gives

$$\exp\left(-\frac{(j-|U|)!e^j\sqrt{j+1}}{(j-|U|)^{j-|U|}}\log n\right) \leqslant \exp\bigl(-(j+1)\log n\bigr) = \frac{1}{n^{j+1}}.$$

Thus, the probability of CheckExistence being incorrect is at most $\frac{1}{n^{j+1}}$.

Now, we bound the probability that BinarySearch performs incorrectly for given arguments $(S_i)_{i=1}^{j}$ and r. To this end, we consider an imaginary procedure BS' that is the same as BinarySearch except that in BS', the repeat loop continues until the size of S_r becomes one. Suppose that the size of S_r becomes one after S_r is halved and updated t times. For $1 \leqslant i \leqslant t$, let X_i be the number of iterations of the repeat loop between the $(i-1)$th and ith updates of S_r. Let v be a vertex of a hyperedge crossing among S_i's that is in the initial S_r. When v is in the $(i-1)$ times updated S_r, the probability that v is chosen as an element of S'_r is at least $\frac{1}{3}$. (The extreme case is when the size of S_r is three.) Thus, X_i follows a geometric distribution with parameter at least $\frac{1}{3}(1-\delta)$. If we let $X = \sum_{i=1}^{t} X_i$, by linearity of expectation,

$$\mathrm{E}[X] \leqslant \frac{3t}{1-\delta}.$$

Thus,

$$\Pr\left[X > \frac{6(j+1)}{1-\delta}\log n\right] \leqslant \Pr\left[X > \left(\frac{2(j+1)\log n}{t}\right)\mathrm{E}[X]\right].$$

Since X_i's are independent, letting $1 + \alpha = \frac{2(j+1)\log n}{t}$, Lemma 4.3 gives

$$\Pr\left[X > \frac{6(j+1)}{1-\delta}\log n\right] \leqslant \exp\left(-\frac{\alpha^2 t}{2(1+\alpha)}\right) \leqslant \exp\bigl(-(j+1)\log n\bigr) = \frac{1}{n^{j+1}}.$$

Thus, the probability of BinarySearch performing incorrectly is at most $\frac{1}{n^{j+1}}$ as it is at most the probability of X being more than $\lceil \frac{6(j+1)}{1-\delta}\log n \rceil$, the number of iterations in BinarySearch.

The number of CheckExistence and BinarySearch calls for finding G_j are at most $2j^2|E(G_j)|+1$ and $j^2|E(G_j)|$, respectively. Thus, in the process of finding G_j, the probability that CheckExistence or BinarySearch incorrectly perform once or more times is at most $\frac{3j^2|E(G_j)|+1}{n^{j+1}} \leqslant \frac{3j^2n^j+1}{n^{j+1}}$, which is $\mathcal{O}(\frac{1}{n})$ as desired. □

Theorem 1.3 follows from Lemmas 4.1 and 4.4. Here, we mention that it is more straightforward to obtain $\mathcal{O}(m\log^2 n)$ algorithm for finding hypergraphs (and hence $\mathcal{O}(m\log^2 n)$ algorithm for finding Fourier coefficients) by querying the oracle $\Theta(\log n)$ times for each cross-detecting query to make the error probability $\mathcal{O}(1/\mathrm{poly}(n))$.

5. Remarks on query and time complexity

We give some remarks on the query and time complexity for finding the Fourier coefficients of k-bounded functions based on GFA. We assume that k is a constant independent of n.

Suppose that we are given a k-bounded function f on $\{0,1\}^n$ with m non-zero Fourier coefficients. To find the Fourier coefficients of f, we first find the hyperedges of the linkage graph G_f. From Theorem 3.3, we have the oracle with one-sided error $\delta = 1-\frac{1}{2^{2k}}$ that gives the answer for a cross-detecting query by using 2^k function evaluations. Since f has m non-zero Fourier coefficients, G_f has at most $2^k m$ hyperedges. For G_f with n vertices and at most $2^k m$ hyperedges, GFA uses $\mathcal{O}(\frac{(2e)^k k^{3.5}}{1-\delta} m\log n)$ cross-detecting queries as shown in the proof of Lemma 4.1. Thus, we can find G_f (with high probability) by using $\mathcal{O}((16e)^k k^{3.5} m\log n)$ function evaluations.

Once G_f is obtained, the Fourier coefficients of f can be found by using $\mathcal{O}(2^k m)$ additional function evaluations from Proposition 2.2 and the arguments after the proposition. Thus, the overall query complexity for finding the Fourier coefficients of f (with high probability) is $\mathcal{O}((16e)^k k^{3.5} m\log n)$. This is $\mathcal{O}(m\log n)$ for constant k and Theorem 1.1 follows. Another important issue in practical applications is the time complexity. From the pseudocode of GFA, we can check that the time complexity of the algorithm for finding the Fourier coefficients is $\mathcal{O}(nm\log n)$ for constant k and it is exponential in k.

6. Some improvement issues

We should note that GFA does not assume the hierarchical property among hyperedges. By using the hierarchical property, we may obtain a more efficient algorithm to find the linkage graphs of k-bounded functions. This improves the query complexity of finding the Fourier coefficients of k-bounded functions in terms of k. In this section, assuming that k is a constant independent of n, we briefly describe an algorithm based on the hierarchical property and minor improvement factors and present its query complexity. We call it *Linkage-Graph Finding Algorithm* (LGFA) as it exploits the properties appearing in the linkage graphs of k-bounded functions (not appearing in arbitrary hypergraphs).

Given a k-bounded function f on $\{0,1\}^n$, LGFA finds the linkage graph G_f with error probability $\mathcal{O}(\frac{1}{n})$. It successively finds the hyperedges of size j for $j=1,\ldots,k$. In the beginning, LGFA finds the hyperedges of size 1 in the same way as in GFA. For $j \geqslant 2$, suppose that the hyperedges of size $j-1$ have been found. Then, for each hyperedge H of size $j-1$, it finds the hyperedges of size j including H. To find such a hyperedge, LGFA finds a vertex v such that $H\cup\{v\}$ is a hyperedge by iteratively testing the existence of a hyperedge crossing among $\mathcal{S}=\{S_1,\ldots,S_j\}$, where $S_1,\ldots,S_{j-1}$ are the singleton sets of $j-1$ vertices in H and S_j is $[n]\setminus H$. More specifically, it tests iteratively $\lceil 2^{2k}(j+1)\log n\rceil$ times whether $\mathfrak{L}^*(f,\mathcal{R},x)\neq 0$ for $\mathcal{R}$ generated from $\Gamma(\mathcal{S})$ and x chosen uniformly at random from $\mathrm{str}(\mathcal{R})$. If all the tests fail, LGFA terminates the process of finding the hyperedges of size j. If $\mathfrak{L}^*(f,\mathcal{R},x)\neq 0$ for some $\mathcal{R}=\{R_1,\ldots,R_j\}$ and x, this means that there is a hyperedge of the form $H\cup\{v\}$ with $v\in R_j$. Due to the following, we may find such a vertex v by binary search on R_j without additional function evaluations, unlike in BinarySearch of GFA.

Lemma 6.1 *(Binary search). Suppose that f is a pseudo-Boolean function on $\{0,1\}^n$, $\mathcal{R}$ is a collection of disjoint subsets of $[n]$, and A is an element of $\mathcal{R}$. If (A_1,A_2) is a partition of A, and $\mathfrak{L}^*(f,\mathcal{R},x)\neq 0$ and $\mathfrak{L}^*(f,\mathcal{R}\cup\{A_1\}\setminus\{A\},x)=0$ for some $x\in\{0,1\}^n$, then $\mathfrak{L}^*(f,\mathcal{R}\cup\{A_2\}\setminus\{A\},x\oplus 1_{A_1})\neq 0$.*

Proof. Let $\mathcal{R}'=\mathcal{R}\cup\{A_1\}\setminus\{A\}$ and $\mathcal{R}''=\mathcal{R}\cup\{A_2\}\setminus\{A\}$. To prove the lemma, it is enough to show

$$\mathfrak{L}^*\big(f,\mathcal{R}'',x\oplus 1_{A_1}\big)=\mathfrak{L}^*(f,\mathcal{R},x)-\mathfrak{L}^*\big(f,\mathcal{R}',x\big).$$

By Lemma 3.1(b),

$$\begin{aligned}\mathfrak{L}^*\big(f,\mathcal{R}'',x\oplus 1_{A_1}\big)&=\mathfrak{L}^*\big(f,\mathcal{R}''\setminus\{A_2\},x\oplus 1_{A_1}\big)-\mathfrak{L}^*\big(f,\mathcal{R}''\setminus\{A_2\},x\oplus 1_{A_1}\oplus 1_{A_2}\big)\\&=\mathfrak{L}^*\big(f,\mathcal{R}''\setminus\{A_2\},x\oplus 1_{A_1}\big)-\mathfrak{L}^*\big(f,\mathcal{R}''\setminus\{A_2\},x\oplus 1_{A}\big)\end{aligned}$$

as $1_{A_1}\oplus 1_{A_2}=1_A$. Since $\mathcal{R}''\setminus\{A_2\}=\mathcal{R}'\setminus\{A_1\}=\mathcal{R}\setminus\{A\}$,

$$\mathfrak{L}^*\big(f,\mathcal{R}'',x\oplus 1_{A_1}\big)=\mathfrak{L}^*\big(f,\mathcal{R}'\setminus\{A_1\},x\oplus 1_{A_1}\big)-\mathfrak{L}^*\big(f,\mathcal{R}\setminus\{A\},x\oplus 1_A\big).$$

By Lemma 3.1(b) again, $\mathfrak{L}^*(f,\mathcal{R}'\setminus\{A_1\},x\oplus 1_{A_1})=\mathfrak{L}^*(f,\mathcal{R}'\setminus\{A_1\},x)-\mathfrak{L}^*(f,\mathcal{R}',x)$ and $\mathfrak{L}^*(f,\mathcal{R}\setminus\{A\},x\oplus 1_A)=\mathfrak{L}^*(f,\mathcal{R}\setminus\{A\},x)-\mathfrak{L}^*(f,\mathcal{R},x)$. Plugging this into the above,

$$\begin{aligned}\mathfrak{L}^*\big(f,\mathcal{R}'',x\oplus 1_{A_1}\big)&=\big(\mathfrak{L}^*\big(f,\mathcal{R}'\setminus\{A_1\},x\big)-\mathfrak{L}^*\big(f,\mathcal{R}',x\big)\big)-\big(\mathfrak{L}^*\big(f,\mathcal{R}\setminus\{A\},x\big)-\mathfrak{L}^*(f,\mathcal{R},x)\big)\\&=\mathfrak{L}^*(f,\mathcal{R},x)-\mathfrak{L}^*\big(f,\mathcal{R}',x\big)\end{aligned}$$

as $\mathfrak{L}^*(f,\mathcal{R}'\setminus\{A_1\},x)=\mathfrak{L}^*(f,\mathcal{R}\setminus\{A\},x)$. □

After finding a hyperedge including H, LGFA repeats the above process to find the other hyperedges that have not been found. If we let W be the set of the vertices v such that $H\cup\{v\}$ is a hyperedge that LGFA has found, this can be done by testing the existence of a hyperedge crossing among $\{S_1,\dots,S_j\}$, where $S_1,\dots,S_{j-1}$ are the singleton sets of $j-1$ vertices in H and S_j is $[n]\setminus(H\cup W)$.

The major feature of LGFA is that LGFA finds the hyperedges of size j by considering only the subsets of size j that contain some hyperedge of size $j-1$ that has been already found. This is based on the hierarchical property in the linkage graphs. Due to this feature, $\mathcal{O}(2^{2k}(j+1)\log n)$ linkage tests for checking the existence of a hyperedge of size j is enough to guarantee that the overall error probability in finding the hyperedges of size j is $\mathcal{O}(\frac{1}{n})$. (Compare this to the iteration number in the repeat loop in CheckExistence of GFA.) If f has m non-zero Fourier coefficients, the number of hyperedges of size j in G_f is at most $\binom{k}{j}m$. Thus, the number of function evaluations by LGFA to find G_f is $\mathcal{O}((16)^k km\log n)$ as

$$\sum_{j=1}^{k}\binom{k}{j}m\big(2^{2k}(j+1)\log n\big)2^k=\mathcal{O}\big((16)^k km\log n\big).$$

If we use LGFA instead of GFA, the query complexity of finding the Fourier coefficients is $\mathcal{O}((16)^k km\log n)$ for a k-bounded function on $\{0,1\}^n$ with m non-zero Fourier coefficients.

7. Lower bounds

In this section, we show the lower bounds for finding the Fourier coefficients of k-bounded functions and for finding hypergraphs using cross-detecting queries.

For finding the Fourier coefficients, denoted by $\mathcal{F}(n,m,k)$ is the set of k-bounded functions on $\{0,1\}^n$ with m non-zero Fourier coefficients. Then, we have the following.

Theorem 7.1. *Suppose that k is a constant independent of n and ε is a positive number bounded above by a constant less than 1. Then, any randomized algorithm that for any $f\in\mathcal{F}(n,m,k)$ finds the Fourier coefficients of f with error probability at most ε requires $\Omega(\frac{m\log n}{\log m})$ function evaluations.*

The same bound was proven by Choi, Jung, and Moon [13] provided that $m\leqslant n^{k-\alpha}$ for any constant $\alpha>0$. (Actually, they showed the bound for finding the linkage graphs of k-bounded functions.) Our result extends the bound for finding the Fourier coefficients to all m.

For finding hypergraphs, a better bound may be obtained even for the oracle of which answers are always correct. In the following, we assume such an oracle for cross-detecting queries and $\mathcal{G}(n,m,k)$ denotes the set of k-bounded hypergraphs on the vertex set $[n]$ with m hyperedges.

Theorem 7.2. *Suppose that k is a constant independent of n and ε is a positive number bounded above by a constant less than 1. Then, any randomized algorithm that for any $G\in\mathcal{G}(n,m,k)$ finds the hyperedges of G with error probability at most ε requires $\Omega(m\log n)$ cross-detecting queries, provided that $m\leqslant n^{k-\alpha}$ for any constant $\alpha>0$.*

The main tool for obtaining the above theorems is Yao's minimax principle [41,31].

Proposition 7.3 *(Yao). Consider a complexity model for computing a function F. Let $R_\varepsilon(F)$ be the minimum complexity over all randomized algorithms that, for all input x, compute $F(x)$ with error probability at most ε. Given a distribution μ on the inputs, let $D_\varepsilon^\mu(F)$ be the minimum complexity over all deterministic algorithms that correctly compute F on a fraction of at least $1-\varepsilon$ of all inputs with respect to μ. Then,*

$$R_\varepsilon(F)=\max_\mu D_\varepsilon^\mu(F).$$

This may be restated as follows in the context of finding Fourier coefficients. We simply write $\mathcal{F}$ for $\mathcal{F}(n,m,k)$.

Corollary 7.4. *Suppose that ε is a real number between 0 and 1. Let $R_\varepsilon[\mathcal{F}]$ be the minimum number of function evaluations by a randomized algorithm that for any $f \in \mathcal{F}$ finds the Fourier coefficients of f with error probability at most ε. Given a distribution μ on $\mathcal{F}$, let $D^\mu_\varepsilon[\mathcal{F}]$ be the minimum number of function evaluations by a deterministic algorithm that, for a function f sampled according to μ, finds the Fourier coefficients of f with error probability at most ε. Then,*

$$R_\varepsilon[\mathcal{F}] = \max_\mu D^\mu_\varepsilon[\mathcal{F}].$$

Now, we prove Theorem 7.1.

Proof of Theorem 7.1. We may assume that $m \geqslant n^\delta$ for some small constant $\delta > 0$, say $\delta = 0.01$, as the previous result in [13] implies the bound in the other range. We consider a subclass $\mathcal{F}'$ of $\mathcal{F}$ that consists of the functions in $\mathcal{F}$ of which non-zero Fourier coefficients are integers between 1 and n. We define μ as the uniform distribution over the functions in $\mathcal{F}'$. When a function f is given according to μ, we will show that any deterministic algorithm requires $\Omega(\frac{m\log n}{\log m})$ function evaluations to find the Fourier coefficients of f with error probability at most ε. Then, Theorem 7.1 follows from Corollary 7.4.

Consider a deterministic algorithm A that takes a pseudo-Boolean function f according to μ as an input and outputs the Fourier coefficients of f. Suppose that A performs at most τ function evaluations. Since a function in $\mathcal{F}'$ has at most $2mn+1$ integer values between $-mn$ and mn, there are at most $(2mn+1)^\tau$ different combinations of function values that A gets from the functions in $\mathcal{F}'$. Also, A is a deterministic algorithm and so the output of A is uniquely determined by a combination of function values. Hence, A has at most $(2mn+1)^\tau$ different outputs. Since the functions in $\mathcal{F}'$ have different sets of Fourier coefficients, the output of A may be correct for at most $(2mn+1)^\tau$ inputs from $\mathcal{F}'$. If we let $M := \sum_{i=0}^{k}\binom{n}{i}$, there are $\binom{M}{m}n^m$ functions in $\mathcal{F}'$. As μ is the uniform distribution over the functions in $\mathcal{F}'$, we have

$$\Pr[A \text{ is correct}] \leqslant \frac{(2mn+1)^\tau}{\binom{M}{m}n^m}.$$

Thus, the probability over μ that A is correct is less than $1-\varepsilon$ unless $\tau \geqslant \frac{\log(1-\varepsilon)\binom{M}{m}n^m}{\log(2mn+1)}$. The desired result follows as $\frac{\log(1-\varepsilon)\binom{M}{m}n^m}{\log(2mn+1)} = \Omega(\frac{m\log n}{\log m})$ if $m \geqslant n^{0.01}$. □

We may prove Theorem 7.2 essentially in the same manner as for Theorem 7.1. The main difference is that a deterministic algorithm for finding hypergraphs gets only two possible answers for a cross-detecting query while the number of hypergraphs in $\mathcal{G}(n,m,k)$ is still $e^{\Omega(m\log n)}$ provided that k is constant and $m \leqslant n^{k-\alpha}$ for any constant $\alpha > 0$. For that reason, more queries are required for the algorithm to correctly find a desired fraction of the hypergraphs in $\mathcal{G}(n,m,k)$. We omit the detail of the proof.

8. Conclusion

In this paper, we proved an almost tight upper bound in query complexity for finding the Fourier coefficients of k-bounded functions for constant k. To this end, we first showed that the problem of finding the Fourier coefficients is reduced to the problem of finding k-bounded hypergraphs with cross-detecting queries under the oracle with one-sided error. Then, we gave a randomized algorithm for the hypergraph finding problem and analyzed it to obtain the desired bound. In particular, we obtained the almost optimal bound even without using the hierarchical property among hyperedges, which is crucially used in the best previous algorithm.

As shown in the previous sections, the query and time complexities of the proposed algorithms are exponential in k. Although the main concern of this paper is the case when k is constant, it would be worth trying to find an algorithm with better query and time complexity for general k.

References

[1] N. Alon, V. Asodi, Learning a hidden subgraph, in: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), 2004, pp. 110–121.
[2] N. Alon, V. Asodi, Learning a hidden subgraph, SIAM J. Discrete Math. 18 (4) (2005) 697–712.
[3] N. Alon, R. Beigel, S. Kasif, S. Rudich, B. Sudakov, Learning a hidden matching, in: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002), 2002, pp 197–206.
[4] N. Alon, R. Beigel, S. Kasif, S. Rudich, B. Sudakov, Learning a hidden matching, SIAM J. Comput. 33 (2) (2004) 487–501.
[5] D. Angluin, J. Chen, Learning a hidden graph using $\mathcal{O}(\log n)$ queries per edge, in: Proceedings of the 17th Annual Conference on Learning Theory (COLT 2004), 2004, 210–223.
[6] D. Angluin, J. Chen, Learning a hidden hypergraph, in: Proceedings of the 18th Annual Conference on Learning Theory (COLT 2005), 2005, 561–575.
[7] D. Angluin, J. Chen, Learning a hidden hypergraph, J. Mach. Learn. Res. 7 (2006) 2215–2236.
[8] R. Beigel, N. Alon, M.S. Apaydin, L. Fortnow, S. Kasif, An optimal procedure for gap closing in whole genome shotgun sequencing, in: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB 2001), 2001, 22–30.
[9] M. Bouvel, V. Grebinski, G. Kucherov, Combinatorial search on graphs motivated by bioinformatics applications: A brief survey, in: The 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), 2005, pp. 16–27.

[10] N.H. Bshouty, J.C. Jackson, C. Tamon, More efficient PAC-learning of DNF with membership queries under the uniform distribution, J. Comput. System Sci. 68 (1) (2004) 205–234.
[11] N.H. Bshouty, C. Tamon, On the Fourier spectrum of monotone functions, J. ACM 43 (4) (1996) 747–770.
[12] H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, Ann. Math. Statist. 23 (1952) 493–509.
[13] S.S. Choi, K. Jung, B.R. Moon, Lower and upper bounds for linkage discovery, IEEE Trans. on Evolutionary Computation 13 (2) (2009) 201–216.
[14] S.S. Choi, J.H. Kim, Optimal query complexity bounds for finding graphs, Artificial Intelligence 174 (9–10) (2010) 551–569.
[15] D.J. Coffin, C.D. Clack, gLINC: Identifying composability using group perturbation, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2006, pp. 1133–1140.
[16] J.S. de Bonet, C.L. Isbell Jr., P. Viola, MIMIC: Finding optima by estimating probability densities, in: Proceedings of the Advances in Neural Information Processing Systems, vol. 9, MIT Press, 1997, pp. 424–430.
[17] V. Feldman, Attribute-efficient and non-adaptive learning of parities and DNF expressions, J. Mach. Learn. Res. 8 (2007) 1431–1460.
[18] W. Fontana, P. Stadler, E. Bornberg-Bauer, T. Griesmacher, I. Hofacker, M. Tacker, P. Tarazona, E. Weinberger, P. Schuster, RNA folding and combinatory landscapes, Phys. Rev. E 47 (3) (1993) 2083–2099.
[19] M.R. Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, Theoret. Comput. Sci. 1 (1976) 237–267.
[20] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.
[21] V. Grebinski, G. Kucherov, Optimal reconstruction of graphs under the additive model, Algorithmica 28 (2000) 104–124.
[22] G.R. Harik, D.E. Goldberg, Learning linkage, in: Foundations of Genetic Algorithms, vol. 4, Morgan Kaufmann, 1997, pp. 247–262.
[23] R.B. Heckendorn, A.H. Wright, Efficient linkage discovery by limited probing, Evol. Comput. 12 (4) (2004) 517–545.
[24] J. Jackson, An efficient membership-query algorithm for learning DNF with respect to the uniform distribution, J. Comput. System Sci. 55 (3) (1997) 42–65.
[25] H. Kargupta, B. Park, Gene expression and fast construction of distributed evolutionary representation, Evol. Comput. 9 (1) (2001) 1–32.
[26] S.A. Kauffman, Adaptation on rugged fitness landscapes, in: D. Stein (Ed.), Lectures in the Sciences of Complexity, Addison Wesley, 1989, pp. 527–618.
[27] S.A. Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford University Press, 1993.
[28] E. Kushilevitz, Y. Mansour, Learning decision trees using the Fourier spectrum, SIAM J. Comput. 22 (6) (1993) 1331–1348.
[29] L.A. Levin, Randomness and non-determinism, J. Symbolic Logic 58 (3) (1993) 1102–1103.
[30] Y. Mansour, Learning Boolean functions via the Fourier transform, in: V. Roychowdhury, K.Y. Siu, A. Orlitsky (Eds.), Theoretical Advances in Neural Computation and Learning, Kluwer Academic, 1994, pp. 391–424.
[31] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[32] H. Mühlenbein, T. Mahnig, FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions, Evol. Comput. 7 (1) (1999) 45–68.
[33] M. Munetomo, D.E. Goldberg, Identifying linkage groups by nonlinearity/non-monotonicity detection, in: Proceedings of the Genetic and Evolutionary Computation Conference, 1999, pp. 433–440.
[34] N. Nisan, M. Szegedy, On the degree of Boolean functions as real polynomials, Comput. Complexity 4 (1994) 301–313.
[35] M. Pelikan, D.E. Goldberg, Hierarchical problem solving by the Bayesian optimization algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2000, pp. 267–274.
[36] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, Linkage problem, distribution estimation, and Bayesian networks, Evol. Comput. 8 (3) (2000) 311–340.
[37] L. Reyzin, N. Srivastava, Learning and verifying graphs using queries with a focus on edge counting, in: Proceedings of the 18th International Conference on Algorithmic Learning Theory (ALT 2007), 2007, pp. 285–297.
[38] J.T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, J. ACM 27 (4) (1980) 701–717.
[39] M.J. Streeter, Upper bounds on the time and space complexity of optimizing additively separable functions, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), 2004, pp. 186–197.
[40] J.L. Walsh, A closed set of orthogonal functions, Amer. J. Math. 55 (1923) 5–24.
[41] A.C. Yao, Probabilistic computations: Toward a unified measure of complexity, in: Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science, 1977, pp. 222–227.